بحث شامل حول البرمجة الكائنية التوجه (OOP) في PHP

إعداد المُهندسه / رغد حمود محمد الحُمري .

الدكتور / إبراهيم الشامي .

بحث شامل حول البرمجة الكائنية التوجه (OOP) فيPHP

مقدمة

البرمجة الكائنية التوجه (OOP) هي أحد الأساليب التي تعتمد على الكائنات (objects) لتنظيم الشيفرة البرمجية وتحقيق مبدأ إعادة الاستخدام. يتم استخدام هذا الأسلوب بشكل واسع في مختلف لغات البرمجة مثلC++, Java, Python ، وكذلك في PHP. من خلالOOP ، يمكن للمبرمجين تنظيم تطبيقاتهم بشكل أكثر مرونة وسهولة في التعديل.

في هذا البحث، سنتناول شرحاً مفصلاً لمفهوم OOP فيPHP ، وسنناقش جميع الأساسيات التي يحتاج المبرمج لفهمها لتطوير تطبيقات باستخدام هذا النمط. سنتعرف على مفاهيم أساسية مثل الفئات(Classes) ، الكائنات(Objects) ، الوراثة(Inheritance) ، التعددية الشكلية (Polymorphism)، والتغليف (Encapsulation) .

الفصل الأول : المفاهيم الأساسية فيOOP

الفئة (Class) : الفئة هي بمثابة قالب أو تصميم للكائنات، حيث تحتوي على الخصائص (attributes)والأساليب .(methods) عندما يتم إنشاء كائن من الفئة، يتم تخصيص قيم الخصائص وتنفيذ الأساليب.

تعريف الفئة:

```php
<?php

class Car {
    // الخصائص
    public $color;
    public $model;

    // الأساليب
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
    }

    // أسلوب لتشغيل السيارة
    public function start() {
        echo "The " . $this->color . " " . $this->model . " car is starting.";
    }
}

// إنشاء كائن من الفئة Car
$myCar = new Car("red", "Toyota");

// استدعاء أسلوب start
$myCar->start();

?>
```

في المثال السابق، Carهي الفئة التي تحتوي على الخصائص color وmodel ، وكذلك الأسلوب start() الذي يطبع رسالة عند استدعائه.

2 .الكائن :(Object) الكائن هو نسخة من الفئة. يتم إنشاء الكائن باستخدام الكلمة المفتاحية newويمنح الخصائص والقيم المعرفة في الفئة.

إنشاء كائن من فئة:

```php
<?php

class Car {
    // الخصائص
    public $color;
    public $model;

    // المنشئ لتعيين القيم عند إنشاء الكائن
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
    }

    // الأسلوب لتشغيل السيارة
    public function start() {
        echo "The " . $this->color . " " . $this->model . " car is starting.";
    }
}

// إنشاء كائن مع تمرير القيم مباشرة
$myCar = new Car('Red', 'Toyota');

// استدعاء الأسلوب
$myCar->start();

?>
```

في المثال السابق، تم إنشاء كائن $myCar من الفئة Car وتعيين بعض القيم للخصائص.

1. التغليف :(Encapsulation) التغليف يعني إخفاء تفاصيل الكائنات من المستخدمين، وذلك من خلال تحديد خصائص وأساليب يمكن الوصول إليها فقط باستخدام واجهات معينة. يتم تحقيق ذلك باستخدام الرؤية(public) ، (private)، (protected).

مثال على التغليف:

```php
<?php

class Car {
    private $engine;  // خاصية خاصة لا يمكن الوصول إليها من خارج الفئة

    // طريقة لتعيين قيمة الخاصية
    public function setEngine($engine) {
        $this->engine = $engine;
    }

    // طريقة للحصول على قيمة الخاصية
    public function getEngine() {
        return $this->engine;
    }
}

// إنشاء كائن من الفئة Car
$myCar = new Car();

// تعيين قيمة للخاصية engine الطريقة باستخدام setEngine
$myCar->setEngine("V8");

// الحصول على قيمة الخاصية engine الطريقة باستخدام getEngine
echo "The engine of the car is: " . $myCar->getEngine();  // ستطبع: The engine of the car is: V8

?>
```

في المثال، يتم إخفاء الخاصية engine باستخدام الكلمة private ولا يمكن الوصول إليها مباشرة من خارج الفئة. يتم استخدام الأساليب setEngine() و getEngine() للوصول إليها.

2 .الوراثة :(Inheritance) الوراثة تسمح بإنشاء فئات جديدة من فئات موجودة، بحيث يمكن للفئة المشتقة أن ترث الخصائص والأساليب من الفئة الأصلية. يمكن توسيع الفئة المشتقة أو تعديل أساليبها.

مثال على الوراثة:

```php
<?php

class Vehicle {
    public $wheels;  // خاصية تمثل عدد العجلات

    // المُنشئ الذي يقبل عدد العجلات
    public function __construct($wheels) {
        $this->wheels = $wheels;
    }
}

class Car extends Vehicle {
    public $color;  // خاصية تمثل لون السيارة

    // مُنشئ الفئة Car، والذي يستدعي مُنشئ الفئة الأصلية Vehicle
    public function __construct($wheels, $color) {
        parent::__construct($wheels);  // استدعاء مُنشئ الفئة الأصلية Vehicle
        $this->color = $color;
    }
}

// إنشاء كائن من الفئة Car
$myCar = new Car(4, 'Red');

// عرض لون السيارة
echo $myCar->color;  // ستطبع: Red

?>
```

في المثال، الفئة Car ترث من الفئة Vehicle ، وتستخدم مُنشئ الفئة الأصلية parent::__construct()لتخصيص قيمة العجلات.

3 . التعددية الشكلية (Polymorphism) : التعددية الشكلية تعني أن الكائنات من فئات مختلفة يمكن أن تستجيب لنفس الأسلوب بطرق مختلفة. يتم تحقيق ذلك من خلال التعريفات المختلفة للأساليب داخل الفئات المشتقة.

مثال على التعددية الشكلية:

```php
<?php

class Animal {
    public function sound() {
        echo "Some sound";
    }
}

class Dog extends Animal {
    public function sound() {
        echo "Bark";
    }
}

class Cat extends Animal {
    public function sound() {
        echo "Meow";
    }
}

$dog = new Dog();
$cat = new Cat();

$dog->sound();   // Bark
$cat->sound();   // Meow


?>
```

في المثال، يتم استخدام نفس الاسم ()sound في كلا من الفئتين Dog و Cat ولكن بتنفيذ مختلف.

الفصل الثالث: أنماط التصميم في OOP

1. نمط التصميم Singleton : يضمن نمط Singleton أنه يمكن إنشاء كائن واحد فقط من فئة معينة. يُستخدم عندما نحتاج إلى فئة يكون لها نسخة واحدة فقط في النظام.

مثال على Singleton :

```php
<?php

class Singleton {
    private static $instance;

    private function __construct() {}

    public static function getInstance() {
        if (self::$instance === null) {
            self::$instance = new Singleton();
        }
        return self::$instance;
    }
}
$singleton = Singleton::getInstance();

?>
```

2 .نمط التصميم :Factory يتيح نمط Factory إنشاء الكائنات بطريقة مرنة بناءً على معايير معينة.

مثال على:Factory

```php
<?php

class VehicleFactory {
    public static function createVehicle($type) {
        if ($type == "Car") {
            return new Car();
        } elseif ($type == "Bike") {
            return new Bike();
}}}
$car = VehicleFactory::createVehicle("Car");

$singleton = Singleton::getInstance();

?>
```

الفصل الرابع: مزايا وعيوب OOP في PHP

المزايا:

إعادة الاستخدام: يمكن إعادة استخدام الكائنات والفئات في تطبيقات أخرى، مما يقلل من تكرار الشيفرة.

إدارة أفضل للمشاريع الكبيرة: معOOP ، يمكن تقسيم المشروع إلى وحدات مستقلة تسهل العمل الجماعي والتطوير المستمر.

قابلية التوسع OOP :يسهل إضافة وظائف جديدة دون التأثير على الشيفرة الحالية.

العيوب:

تعقيد في البداية: يمكن أن تكون OOP معقدة للمبرمجين الجدد في البداية، خاصة مع استخدام المفاهيم المتقدمة مثل الوراثة والتعددية الشكلية.

استهلاك الموارد: في بعض الحالات، قد تستهلك OOP موارد أكثر مقارنة بالبرمجة الإجرائية البسيطة.

الخاتمة

البرمجة الكائنية التوجه هي أسلوب قوي ومرن لتنظيم الشيفرة البرمجية في .PHP. من خلال الفئات والكائنات، يمكن للمطورين كتابة برامج مرنة وقابلة للصيانة. تتضمن OOP العديد من المفاهيم الأساسية مثل التغليف، الوراثة، والتعددية الشكلية، وهي توفر العديد من المزايا لتطوير البرمجيات بشكل مستدام.